Contents

Custom CA Integration using Kubernetes CSR

6 minute read page test

Part 1: Using Kubernetes CA

- Deploying Istio with Kubernetes CA
- Verify that the certificates installed are correct
- Cleanup Part 1

Part 2: Using Custom CA

Deploy Custom CA controller in the Kubernetes cluster
Load the CA root certificate into a secret that istiod can access
Deploying Istio
Verify that Custom CA certificates installed are correct
Cleanup Part 2
Reasons to use this feature
See also

This feature is actively in development and is considered experimental.

This feature requires Kubernetes version >= 1.18.

This task shows how to provision Workload Certificates using a custom certificate authority that integrates with the Kubernetes CSR API. This feature leverages Chiron, a lightweight component linked with Istiod that signs certificates using the Kubernetes CSR API.

This task is split into two parts. The first part demonstrates how to use the Kubernetes CA itself to sign workload certificates. The second part demonstrates how to use a custom CA that integrates with the Kubernetes CSR API to sign your certificates.

Part 1: Using Kubernetes CA

Note that this example should only be used for basic evaluation. The use of the `kubernetes.io/legacy-unknown` signer is NOT recommended in production environments.

Deploying Istio with Kubernetes CA

1. Deploy Istio on the cluster using `istioctl` with the following configuration.

```
$ cat <<EOF > ./istio.yaml
  apiVersion: install.istio.io/v1alpha1
  kind: IstioOperator
  spec:
    components:
      pilot:
        k8s:
          env:
          # Indicate to Istiod that we use a Custom Certificate Author
ity
          - name: EXTERNAL_CA
            value: ISTIOD_RA_KUBERNETES_API
          # Tells Istiod to use the Kubernetes legacy CA Signer
          - name: K8S_SIGNER
            value: kubernetes.io/legacy-unknown
  EOF
$ istioctl install --set profile=demo -f ./istio.yaml
```

2. Deploy the `bookinfo` sample application in the bookinfo namespace. Ensure that the following commands are executed in the Istio root directory.

```
$ kubectl create ns bookinfo
$ kubectl apply -f <(istioctl kube-inject -f samples/bookinfo/platform
/kube/bookinfo.yaml) -n bookinfo
```

Verify that the certificates installed are correct

When the workloads are deployed, above, they send CSR Requests to Istiod which forwards them to the Kubernetes CA for signing. If all goes well, the signed certificates are sent

back to the workloads where they are then installed. To verify that they have been signed by the Kubernetes CA, you need to first extract the signed certificates.

1. Dump all pods running in the namespace.

   ```
   $ kubectl get pods -n bookinfo
   ```

   Pick any one of the running pods for the next step.

2. Get the certificate chain and CA root certificate used by the Istio proxies for mTLS.

   ```
   $ istioctl pc secret <pod-name> -o json > proxy_secret
   ```

   The proxy_secret json file contains the CA root certificate for mTLS in the `trustedCA` field. Note that this certificate is

base64 encoded.

3. The certificate used by the Kubernetes CA (specifically the `kubernetes.io/legacy-unknown` signer) is loaded onto the secret associated with every service account in the bookinfo namespace.

```
$ kubectl get secrets -n bookinfo
```

Pick a secret-name that is associated with any of the service-accounts. These have a “token” in their name.

```
$ kubectl get secrets -n bookinfo <secret-name> -o json
```

The `ca.crt` field in the output contains the base64 encoded Kubernetes CA certificate.

4. Compare the `ca.cert` obtained in the previous step with the contents of the `TrustedCA` field in the step before. These two should be the same.
5. (Optional) Follow the rest of the steps in the `bookinfo example` to ensure that communication between services is working as expected.

Cleanup Part 1

- Remove the `istio-system` and `bookinfo` namespaces:

```
$ kubectl delete ns istio-system
$ kubectl delete ns bookinfo
```

Part 2: Using Custom CA

This assumes that the custom CA implements a controller that has the necessary permissions to read and sign Kubernetes CSR Requests. Refer to the Kubernetes CSR documentation for more details. Note that the steps below are dependent on an external-source and may change.

Deploy Custom CA controller in the Kubernetes cluster

1. For this example, we use an open-source Certificate Authority

implementation. This code builds a controller that reads the CSR resources on the Kubernetes cluster and creates certificates using local keys. Follow the instructions on the page to:

   1. Build the Certificate-Controller docker image
   2. Upload the image to a Docker Registry
   3. Generate the Kubernetes manifest to deploy it

2. Deploy the Kubernetes manifest generated in the previous step on your local cluster in the signer-ca-system namespace.

```
$ kubectl apply -f local-ca.yaml
```

Ensure that all the services are running.

```
$ kubectl get services -n signer-ca-system
  NAME                                                TYPE        CLUSTER-I
P   EXTERNAL-IP   PORT(S)    AGE
  signer-ca-controller-manager-metrics-service   ClusterIP   10.8.9.25
    none        8443/TCP   72s
```

3. Get the public key of the CA. This is encoded in the secret “signer-ca-*” in the signer-ca-system namespace.

```
$ kubectl get secrets signer-ca-5hff5h74hm -n signer-ca-system -o json
```

The `tls.crt` field contains the base64 encoded public key file. Record this for future use.

Load the CA root certificate into

a secret that istiod can access

1. Load the secret into the istiod namespace.

```
$ cat <<EOF > ./external-ca-secret.yaml
  apiVersion: v1
  kind: Secret
  metadata:
    name: external-ca-cert
    namespace: istio-system
  data:
  root-cert.pem: <tls.cert from the step above>
  EOF
$ kubectl apply -f external-ca-secret.yaml
```

This step is necessary for Istio to verify that the workload certificates have been signed by the correct certificate authority and to add the root-cert to the trust bundle for

mTLS to work.

Deploying Istio

1. Deploy Istio on the cluster using `istioctl` with the following configuration.

```
$ cat <<EOF > ./istio.yaml
apiVersion: install.istio.io/v1alpha1
kind: IstioOperator
spec:
  components:
    pilot:
      k8s:
        env:
          # Indicate to Istiod that we use an external signer
          - name: EXTERNAL_CA
```

```
            value: ISTIOD_RA_KUBERNETES_API
          # Indicate to Istiod the external k8s Signer Name
          - name: K8S_SIGNER
            value: example.com/foo
        overlays:
          # Amend ClusterRole to add permission for istiod to approve
certificate signing by custom signer
          - kind: ClusterRole
            name: istiod-clusterrole-istio-system
            patches:
              - path: rules[-1]
                value: |
                  apiGroups:
                  - certificates.k8s.io
                  resourceNames:
                  - example.com/foo
                  resources:
                  - signers
                  verbs:
                  - approve
          - kind: Deployment
            name: istiod
```

```
            patches:
              - path: spec.template.spec.containers[0].volumeMounts[-1
]
                value: |
                  # Mount external CA certificate into Istiod
                  name: external-ca-cert
                  mountPath: /etc/external-ca-cert
                  readOnly: true
              - path: spec.template.spec.volumes[-1]
                value: |
                  name: external-ca-cert
                  secret:
                    secretName: external-ca-cert
                    optional: true
EOF
$ istioctl install --set profile=demo -f ./istio.yaml
```

2. Deploy the `bookinfo` sample application in the bookinfo namespace.

```
$ kubectl create ns bookinfo
$ kubectl apply -f <(istioctl kube-inject -f samples/bookinfo/platform
/kube/bookinfo.yaml) -n bookinfo
```

Verify that Custom CA certificates installed are correct

When the workloads are deployed, above, they send CSR Requests to Istiod which forwards them to the Kubernetes CA for signing. If all goes well, the signed certificates are sent back to the workloads where they are then installed. To verify that they have indeed been signed by the Kubernetes CA, you need to first extract the signed certificates.

1. Dump all pods running in the namespace.

```
$ kubectl get pods -n bookinfo
```

   Pick any of the running pods for the next step.

2. Get the certificate chain and CA root certificate used by the Istio proxies for mTLS.

```
$ istioctl pc secret <pod-name> -n bookinfo -o json > proxy_secret
```

   The `proxy_secret` json file contains the CA root certificate for mTLS in the `trustedCA` field. Note that this certificate is base64 encoded.

3. Compare the CA root certificate obtained in the step above with “root-cert.pem” value in external-ca-cert. These two

should be the same.

4. (Optional) Follow the rest of the steps in the bookinfo example to ensure that communication between services is working as expected.

Cleanup Part 2

- Remove the `istio-system` and `bookinfo` namespaces:

```
$ kubectl delete ns istio-system
$ kubectl delete ns bookinfo
```

Reasons to use this feature

- Added Security - Unlike `plugin-ca-cert` or the default `self-signed` option, enabling this feature means that the CA private keys need not be present in the Kubernetes cluster.
- Custom CA Integration - By specifying a Signer name in the Kubernetes CSR Request, this feature allows Istio to integrate with custom Certificate Authorities using the Kubernetes CSR API interface. This does require the custom CA to implement a Kubernetes controller to watch the `CertificateSigningRequest` and `Certificate` Resources and act on them.